Secure Evaluation of Private Linear Branching Programs with Medical Applications

(Full Version)[⋆]

Mauro Barni[1], Pierluigi Failla[1], Vladimir Kolesnikov[2], Riccardo Lazzeretti[1], Ahmad-Reza Sadeghi[3], and Thomas Schneider[3]

[1] Department of Information Engineering, University of Siena, Italy
`barni@dii.unisi.it`,`{pierluigi.failla,lazzaro79}@gmail.com`[⋆⋆]

[2] Bell Laboratories, 600 Mountain Ave. Murray Hill, NJ 07974, USA
`kolesnikov@research.bell-labs.com`

[3] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
`{ahmad.sadeghi,thomas.schneider}@trust.rub.de`[⋆⋆⋆]

Abstract. Diagnostic and classification algorithms play an important role in data analysis, with applications in areas such as health care, fault diagnostics, or benchmarking. Branching programs (BP) is a popular representation model for describing the underlying classification/diagnostics algorithms. Typical application scenarios involve a client who provides data and a service provider (server) whose diagnostic program is run on client's data. Both parties need to keep their inputs private.
We present new, more efficient privacy-protecting protocols for remote evaluation of such classification/diagnostic programs. In addition to efficiency improvements, we generalize previous solutions – we securely evaluate private linear branching programs (LBP), a useful generalization of BP that we introduce. We show practicality of our solutions: we apply our protocols to the privacy-preserving classification of medical ElectroCardioGram (ECG) signals and present implementation results. Finally, we discover and fix a subtle security weakness of the most recent remote diagnostic proposal, which allowed malicious clients to learn partial information about the program.

1 Introduction

Classification and diagnostic programs are very useful tools for automatic data analysis with respect to specific properties. They are deployed for various applications, from spam filters [DCDZ05], remote software fault diagnostics [HRD+07] to medical diagnostic expert systems [RGI05]. The health-care industry is moving faster than ever toward technologies that offer personalized online self-service, medical error reduction, consumer data mining and more (e.g., [Goo09]). Such technologies have the potential of revolutionizing the way medical data is stored, processed, delivered, and made available in an ubiquitous and seamless way to millions of users all over the world.

Typical application scenarios in this context concern two (remote) parties, a user or data provider (client) and a service provider (server) who usually owns the diagnostic software that will run on the client's data and output classification/diagnostic results.

In this framework, however, a central problem is the protection of privacy of both parties. On the one hand, the user's data might be sensitive and security-critical (e.g., electronic patient records in health care, passwords and other secret credentials in remote software diagnostics, trade- and work-flow information in benchmarking of enterprises). On the other hand, the service provider, who owns the diagnostic software, may not be willing to disclose the underlying algorithms and the corresponding optimized parameters (e.g., because they represent intellectual property).

Secure function evaluation with private functions [SYY99,Pin02,KS08b,SS08] is one way to realize the above scenarios, when the underlying private algorithms are represented as circuits. However, as we elaborate in the discussion on related work, in some applications, such as diagnostics, it is most natural and efficient

[⋆] This is the full version of the ESORICS 2009 paper [BFK+09].

[⋆⋆] Supported by EU FP6 project SPEED and MIUR project 2007JXH7ET.

[⋆⋆⋆] Supported by EU FP6 project SPEED and EU FP7 project CACE.

to represent the function as a decision graph or a Branching Program (BP). At a high level, BPs consist of different types of nodes — decision nodes and classification nodes. Based on the inputs and certain decision parameters such as thresholds (that are often the result of learning processes), the algorithm branches among the decision nodes until it reaches the corresponding classification node (which represents a leaf node in the decision tree).

In this work, we consider applications that benefit from the BP representation, such as our motivating application, classification of medical ElectroCardioGram (ECG) signals. In the remainder of the paper, we concentrate on the BP approach (including discussion of related work).

Related Work. There is a number of fundamental works, e.g. Kilian [Kil88], that rely on Branching Programs (BP) "under the hood". These are general feasibility results that do not attempt to achieve high efficiency for concrete problems. The goals and results of these works and ours are different. We do not directly compare their performance to ours; instead, we compare our work with previously-best approaches that are applicable to our setting (see below).

Recently, very interesting BP-based crypto-computing protocols were proposed by Ishai and Paskin [IP07] (and later slightly improved by Lipmaa [Lip08] who also presented a variety of applications). In their setting, the server evaluates his program on client's encrypted data. The novelty of the approach of [IP07] is that the communication and client's computation depend on the length (or depth) of BP, and are *independent* of the size of BP. This allows for significant savings in cases of "wide" BP. However, the protocol requires computationally expensive operations on homomorphically encrypted ciphertexts for each node of the BP. Further, the server's computation still depends on the size of BP. The savings achieved by these protocols are not significant in our setting (in applications we are considering, BPs are not wide), and the cost of employed homomorphic encryption operation outweighs the benefit.

Most relevant for this work is the sequence of works [KJGB06,BPSW07,Sch08], where the authors consider problems similar to ours, and are specifically concerned with concrete performance of the resulting protocols. Kruger et al. [KJGB06] observed that some functions are more succinctly represented by Ordered Binary Decision Diagrams (OBDD), and proposed a natural extension of the garbled circuit method which allows secure evaluation of (publicly known) OBDDs. As in the garbled circuit approach, the client receives garblings of his inputs, and is blindly evaluating a garbled OBDD to receive a garbling of the output, which is then opened. Brickell et al. [BPSW07] further extended this approach and considered evaluation of private BPs. They also consider a more complex decision procedure at the nodes of BP (based on the result of integer comparison). The solution of [BPSW07] is especially suited for remote diagnostics, their motivating application.

In the above two approaches the communication complexity depends linearly on the size of the BP, as the size of the garbled BP is linear in the size of the BP. While the computational complexity for the client remains asymptotically the same as in the crypto-computing protocols of [IP07] (linear in the length of the evaluation path), the computational cost is substantially smaller (especially for the server), as only symmetric crypto operations need to be applied to the nodes of the BP. In [Sch08] an extension of the protocol of [KJGB06] for secure evaluation of private OBDDs based on efficient selection blocks [KS08b] was proposed. In our work, we generalize, unify, extend, and improve efficiency of the above three protocols [KJGB06,BPSW07,Sch08].

In addition to circuits and BPs, other (secure) classification methods have been considered, such as those based on neural networks [CL01,OPB07,PCB$^+$08,SS08]. In our work, we concentrate on the BP representation.

Our Contribution and Outline. Our main contribution is a new more efficient modular protocol for secure evaluation of a class of diagnostics/classification problems, which are naturally computed by (a generalization of) decision trees (§3). We work in the semi-honest model, but explain how our protocols can be efficiently secured against malicious adversaries (§3.6). We improve on the previously proposed solutions in several ways. Firstly, we consider a more general problem. It turns out, our motivating example — ECG classification — as well as a variety of other applications, benefit from a natural generalization of Branching Programs (BP) and decision trees, commonly considered before. We introduce and justify *Linear Branching Programs* (LBP) (§3.1), and show how to evaluate them efficiently. Secondly, we fine-tune the performance. We propose several new tricks (for example, we show how to avoid inclusion of classification nodes in the encrypted program). We also employ performance-improving techniques which were used in a variety of areas of secure computation. This results in significant performance improvements over previous work, even for evaluation of previously considered BPs. A detailed performance comparison is presented in §3.5. Further, in §4, we discover and fix a subtle vulnerability in the recent and very efficient variant of the protocol for secure BP evaluation [BPSW07] and secure classifier learning [BS09]. Finally, we apply our protocols to the privacy-preserving classification of medical ElectroCardioGram (ECG) signals (§5). We implemented our solution; our experimental results show the practical suitability of our protocols (§6).

2 Preliminaries

In our protocols we combine several standard cryptographic tools (additively homomorphic encryption, oblivious transfer, and garbled circuits) which we summarize in §2.1. Readers familiar with these tools can safely skip §2.1 and continue reading our notational conventions in §2.2.

We denote the symmetric (asymmetric) security parameter with t (T). Recommended sizes for short-term security are $t = 80, T = 1248$ [GQ09].

2.1 Cryptographic Tools

Homomorphic Encryption (HE). We use a semantically secure additively homomorphic public-key encryption scheme. In an additively homomorphic cryptosystem, given encryptions $[\![a]\!]$ and $[\![b]\!]$, an encryption $[\![a+b]\!]$ can be computed as $[\![a+b]\!] = [\![a]\!][\![b]\!]$, where all operations are performed in the corresponding plaintext or ciphertext structure. From this property follows, that multiplication of an encryption $[\![a]\!]$ with a constant c can be computed efficiently as $[\![c \cdot a]\!] = [\![a]\!]^c$ (e.g., with the square-and-multiply method). As instantiation we use the Paillier cryptosystem [Pai99,DJ01] which has plaintext space $\mathbb{Z}_N$ and ciphertext space $\mathbb{Z}^*_{N^2}$, where N is a T-bit RSA modulus. This scheme is semantically secure under the decisional composite residuosity assumption (DCRA). For details on the encryption and decryption function we refer to [DJ01].

Parallel Oblivious Transfer (OT). Parallel 1-out-of-2 Oblivious Transfer for m bitstrings of bitlength ℓ, denoted as OT^m_ℓ, is a two-party protocol as shown in Fig. 1. $\mathcal{S}$ inputs m pairs of ℓ-bit strings $S_i = \langle s^0_i, s^1_i \rangle$ for $i = 1, .., m$ with $s^0_i, s^1_i \in \{0,1\}^\ell$. $\mathcal{C}$ inputs m choice bits $b_i \in \{0,1\}$. At the end of the protocol, $\mathcal{C}$ learns $s^{b_i}_i$, but nothing about $s^{1-b_i}_i$ whereas $\mathcal{S}$ learns nothing about b_i. We use OT^m_ℓ as a black-box primitive in our

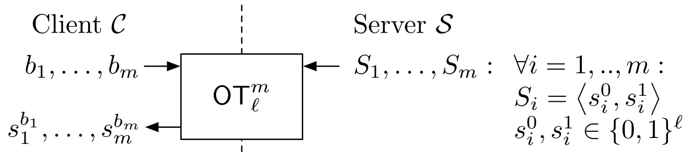
Client $\mathcal{C}$
Server $\mathcal{S}$
$b_1, \ldots, b_m$
OT^m_ℓ
$s^{b_1}_1, \ldots, s^{b_m}_m$
$S_1, \ldots, S_m$: $\forall i = 1, .., m$:
$S_i = \langle s^0_i, s^1_i \rangle$
$s^0_i, s^1_i \in \{0,1\}^\ell$

Fig. 1. OT^m_ℓ - Parallel Oblivious Transfer

constructions. It can be instantiated efficiently with different protocols [NP01,AIR01,Lip03,IKNP03]. For example the protocol of [NP01] implemented over a suitably chosen elliptic curve has asymptotic communication complexity $m(4t+2r)$ and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the random oracle model. The protocol of [AIR01] implemented over a suitably chosen elliptic curve has asymptotic communication complexity $m(12t)$ and is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the standard model. Extensions of [IKNP03] can be used to reduce the number of computationally expensive public-key operations to be independent of m. We omit the parameters m or ℓ if they are clear from the context.

Garbled Circuit (GC). Yao's Garbled Circuit approach [Yao86], excellently presented in [LP04], is the most efficient method for secure evaluation of a boolean circuit C. We summarize its ideas in the following. First, the circuit **constructor** (server $\mathcal{S}$), creates a *garbled circuit* $\widetilde{C}$ with algorithm $\mathsf{CreateGC}$: for each wire W_i of the circuit, he randomly chooses a *complementary garbled value* $\widetilde{W}_i = \langle \widetilde{w}_i^0, \widetilde{w}_i^1 \rangle$ consisting of two secrets, $\widetilde{w}_i^0$ and $\widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the *garbled value* of W_i's value j. (Note: $\widetilde{w}_i^j$ does not reveal j.) Further, for each gate G_i, $\mathcal{S}$ creates and sends to the **evaluator** (client $\mathcal{C}$) a *garbled table* $\widetilde{T}_i$ with the following property: given a set of garbled values of G_i's inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding G_i's output, and nothing else. Then garbled values corresponding to $\mathcal{C}$'s inputs x_j are (obliviously) transferred to $\mathcal{C}$ with a parallel oblivious transfer protocol OT: $\mathcal{S}$ inputs complementary garbled values $\widetilde{W}_j$ into the protocol; $\mathcal{C}$ inputs x_j and obtains $\widetilde{w}_j^{x_j}$ as outputs. Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ with algorithm EvalGC to obtain the garbled output simply by evaluating the garbled circuit gate by gate, using the garbled tables $\widetilde{T}_i$. Correctness of GC follows from method of construction of garbled tables $\widetilde{T}_i$. As in [BPSW07] we use the GC protocol as a conditional oblivious transfer protocol where we do not provide a translation from the garbled output values to their plain values to $\mathcal{C}$, i.e., $\mathcal{C}$ obtains one of two garbled values which can be used as key in subsequent protocols but does not know to which value this key corresponds.

Implementation Details. A *point-and-permute technique* can be used to speed up the implementation of the GC protocol [MNPS04]: The garbled values $\widetilde{w}_i = \langle k_i, \pi_i \rangle$ consist of a symmetric key $k_i \in \{0,1\}^t$ and $\pi_i \in \{0,1\}$ is a random permutation bit. The permutation bit π_i is used to select the right table entry for decryption with the key k_i. Extensions of [KS08a] to "free XOR" gates can be used to further improve performance of GC.

2.2 Notation

Number Representation. In the following, a *(signed) ℓ-bit integer* x^ℓ is represented as one bit for the sign, $sign(x^\ell)$, and $\ell - 1$ bits for the magnitude, $abs(x^\ell)$, i.e., $-2^{\ell-1} < x^\ell < +2^{\ell-1}$. This allows *sign-magnitude representation* of numbers in a circuit, i.e., one bit for the sign and $\ell - 1$ bits for the magnitude. For homomorphic encryptions we use *ring representation*, i.e., x^ℓ with $2^\ell \le N$ is mapped into an element of the plaintext group $\mathbb{Z}_N$ using $m(x^\ell) = \begin{cases} x^\ell, & \text{if } x^\ell \ge 0 \\ N + x^\ell, & \text{if } x^\ell < 0 \end{cases}$.

Homomophic Encryption. $\mathsf{Gen}(1^T)$ denotes the key generation algorithm of the Paillier cryptosystem [Pai99,DJ01] which, on input the asymmetric security parameter T, outputs secret key $sk_{\mathcal{C}}$ and public key $pk_{\mathcal{C}} = N$ to $\mathcal{C}$, where N is a T-bit RSA modulus. $[\![x^\ell]\!]$ denotes the encryption of an ℓ-bit message $x^\ell \in \mathbb{Z}_N$ (we assume $\ell < T$) with public key $pk_{\mathcal{C}}$.

Garbled Objects. Objects overlined with a tilde symbol denote garbled objects: Intuitively, $\mathcal{C}$ cannot infer the real value i from a garbled value $\widetilde{w}^i$, but can use garbled values to evaluate a garbled circuit $\widetilde{C}$ or a garbled LBP $\widetilde{\mathcal{L}}$. Capital letters $\widetilde{W}$ denote complementary garbled values consisting of two garbled values $\langle \widetilde{w}^0, \widetilde{w}^1 \rangle$ for which we use the corresponding small letters. We group together multiple garbled values to a *garbled ℓ-bit value* $\widetilde{\mathbf{w}}^\ell$ (small, bold letter) which consists of ℓ garbled values $\widetilde{w}_1, \ldots, \widetilde{w}_\ell$. Analogously, a *complementary garbled ℓ-bit value* $\widetilde{\mathbf{W}}^\ell$ (capital, bold letter) consists of ℓ complementary garbled values $\widetilde{W}_1, \ldots, \widetilde{W}_\ell$.

3 Secure Evaluation of Private Linear Branching Programs

After formally defining Linear Branching Programs (LBP) in §3.1, we present two protocols for secure evaluation of private LBPs. We decompose our protocols into different building blocks similar to the protocol of [BPSW07] and show how to instantiate them more efficiently than in [BPSW07].

The protocols for secure evaluation of private LBPs are executed between a server $\mathcal{S}$ in possession of a private LBP, and a client $\mathcal{C}$ in possession of data, called **attribute vector**. Let z be the number of nodes in the LBP, and n be the number of attributes in the attribute vector.

As in most practical scenarios n is significantly larger than z, the protocol of [BPSW07] is optimized for this case. In particular, the size of our securely transformed LBP depends linearly on z but is independent of n.

In contrast to [BPSW07], our solutions do not reveal the total number z of nodes of the LBP, but only its number of decision nodes d for efficiency improvements. In particular, the size of our securely transformed LBP depends linearly on d which is smaller than z by up to a factor of two.

3.1 Linear Branching Programs (LBP)

First, we formally define the notion of linear branching programs. We do so by generalizing the BP definition used in [BPSW07]. We note that BPs – and hence also LBPs – generalize binary classification or decision trees and Ordered Binary Decision Diagrams (OBDDs) used in [KJGB06,Sch08].

Definition 1 (Linear Branching Program). *Let* $\mathbf{x}^\ell = x_1^\ell, .., x_n^\ell$ *be the* **attribute vector** *of signed* ℓ*-bit integer values. A binary* **Linear Branching Program (LBP)** $\mathcal{L}$ *is a triple* $\langle\{P_1, .., P_z\}, \mathit{Left}, \mathit{Right}\rangle$*. The first element is a set of* z *nodes consisting of* d **decision nodes** $P_1, .., P_d$ *followed by* $z - d$ **classification nodes** $P_{d+1}, .., P_z$*.*
Decision nodes P_i*,* $1 \le i \le d$ *are the internal nodes of the LBP. Each* $P_i := \left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$ *is a pair, where* $\mathbf{a_i^\ell} = \left\langle a_{i,1}^\ell, .., a_{i,n}^\ell \right\rangle$ *is the* **linear combination vector** *consisting of* n *signed* ℓ*-bit integer values and* $t_i^{\ell'}$ *is the signed* ℓ'*-bit integer* **threshold** *value with which* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell = \sum_{j=1}^n a_{i,j}^\ell x_j^\ell$ *is compared in this node.* $\mathit{Left}(i)$ *is the index of the next node if* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell \le t_i^{\ell'}$*;* $\mathit{Right}(i)$ *is the index of the next node if* $\mathbf{a_i^\ell} \circ \mathbf{x}^\ell > t_i^{\ell'}$*. Functions* $\mathit{Left}()$ *and* $\mathit{Right}()$ *are such that the resulting directed graph is acyclic.*
Classification nodes $P_j := \langle c_j \rangle$*,* $d < j \le z$ *are the leaf nodes of the LBP consisting of a single classification label* c_j *each.*

To evaluate the LBP $\mathcal{L}$ on attribute vector $\mathbf{x}^\ell$, start with the first decision node P_1. If $\mathbf{a_1^\ell} \circ \mathbf{x}^\ell \le t_1^{\ell'}$, move to node $\mathit{Left}(1)$, else to $\mathit{Right}(1)$. Repeat this process recursively (with corresponding $\mathbf{a_i^\ell}$ and $t_i^{\ell'}$), until reaching one of the classification nodes and obtaining the classification $c = \mathcal{L}(\mathbf{x}^\ell)$.

In the general case of LBPs, the bit-length ℓ' of the threshold values $t_i^{\ell'}$ has to be chosen according to the maximum value of linear combinations:

$$abs(\mathbf{a_i^\ell} \circ \mathbf{x}^\ell) = abs(\sum_{j=1}^n a_{i,j}^\ell x_j^\ell) \le \sum_{j=1}^n 2^{2(\ell-1)} = n2^{2(\ell-1)}$$
$$\Rightarrow \ell' = 1 + \lceil \log_2(n2^{2(\ell-1)}) \rceil = 2\ell + \lceil \log_2 n \rceil - 1. \tag{1}$$

As noted above, LBPs can be seen as a generalization of previous representations:

- **Branching Programs (BP)** as used in [BPSW07] are a special case of LBPs. In a BP, in each decision node P_i the α_i-th input $x_{\alpha_i}^\ell$ is compared with the threshold value $t_i^{\ell'}$, where $\alpha_i \in \{0, .., n\}$ is a private index. In this case, the linear combination vector $\mathbf{a_i^\ell}$ of the LBP decision node degrades to a **selection vector** $\mathbf{a_i} = \langle a_{i,1}, .., a_{i,n} \rangle$, with exactly one entry $a_{i,\alpha_i} = 1$ and all other entries $a_{i,j \ne \alpha_i} = 0$. The bit-length of the threshold values $t_i^{\ell'}$ is set to $\ell' = \ell$.
- **Ordered Binary Decision Diagrams (OBDD)** as used in [KJGB06,Sch08] are a special case of BPs with bit inputs ($\ell = 1$) and exactly two classification nodes ($P_{z-1} = \langle 0 \rangle$ and $P_z = \langle 1 \rangle$).

3.2 Protocol Overview

We start with a high-level overview of our protocol for secure evaluation of private linear branching programs. We then fill in the technical details and outline the differences and improvements of our protocol over previous work in the following sections.

Our protocol SecureEvalPrivateLBP, its main building blocks, and the data and communication flows are shown in Fig. 2. The client $\mathcal{C}$ receives an attribute vector $\mathbf{x}^\ell = \{x_1^\ell, \ldots, x_n^\ell\}$ as input, and the server $\mathcal{S}$ receives a linear branching program $\mathcal{L}$. Upon completion of the protocol, $\mathcal{C}$ outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$, and $\mathcal{S}$ learns nothing. Of course, both $\mathcal{C}$ and $\mathcal{S}$ wish to keep their inputs private. Protocol SecureEvalPrivateLBP is naturally decomposed into the following three phases (cf. Fig. 2).

CreateGarbledLBP. In this phase, $\mathcal{S}$ creates a garbled version of the LBP $\mathcal{L}$. This is done similarly to the garbled-circuit-based previous approaches [BPSW07,KJGB06]. The idea is to randomly permute the LBP, encrypt the pointers on the left and right successor, and garble the nodes, so that the evaluator is unable to deviate from the evaluation path defined by his input.

The novelty of our solution is that each node transition is based on the oblivious comparison of a *linear combination of inputs with a node-specific threshold*. Thus, CreateGarbledLBP additionally processes (and modifies) these values and passes them to the next phase. CreateGarbledLBP can be entirely precomputed by $\mathcal{S}$.

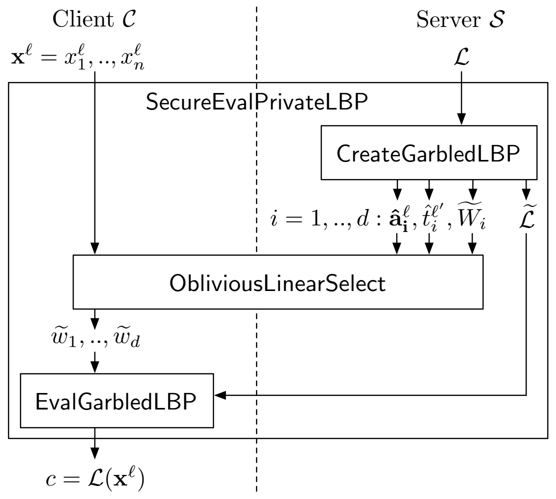
Client $\mathcal{C}$
Server $\mathcal{S}$
$\mathbf{x}^\ell = x_1^\ell,..,x_n^\ell$
$\mathcal{L}$
SecureEvalPrivateLBP
CreateGarbledLBP
$i=1,..,d : \hat{\mathbf{a}}_\mathbf{i}^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
$\widetilde{\mathcal{L}}$
ObliviousLinearSelect
$\widetilde{w}_1,..,\widetilde{w}_d$
EvalGarbledLBP
$c = \mathcal{L}(\mathbf{x}^\ell)$

Fig. 2. Secure Evaluation of Private Linear Branching Programs - Structural Overview

ObliviousLinearSelect. In this phase, $\mathcal{C}$ obliviously obtains the garbled values $\widetilde{w}_1,..,\widetilde{w}_d$ which correspond to the outcome of the comparisons of the linear combination of the attribute vector with the threshold for each garbled node. These garbled values will then be used to evaluate the garbled LBP in the next phase. Making analogy to Yao's garbled circuit (GC), this phase is the equivalent of the GC evaluator receiving the wire secrets corresponding to his inputs. In our protocol, this stage is more complicated, since the secrets are transferred based on secret conditions.

EvalGarbledLBP. This phase is equivalent to Yao's GC evaluation. Here, $\mathcal{C}$ receives the garbled LBP $\widetilde{\mathcal{L}}$ from $\mathcal{S}$, and evaluates it. EvalGarbledLBP additionally gets the garbled values $\widetilde{w}_1,..,\widetilde{w}_d$ output by ObliviousLinearSelect as inputs and outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$.

3.3 Our Building Blocks

Phase I (offline): CreateGarbledLBP. In this pre-computation phase, $\mathcal{S}$ generates a garbled version $\widetilde{\mathcal{L}}$ of the private branching program $\mathcal{L}$. CreateGarbledLBP is presented in Algorithm 1.

Algorithm CreateGarbledLBP converts the nodes P_i of $\mathcal{L}$ into garbled nodes $\widetilde{P}_{\tilde{i}}$ in $\widetilde{\mathcal{L}}$, as follows. First, we associate a randomly chosen key Δ_i with each node P_i. We use Δ_i (with other keys, see below) for encryption of P_i's data. Each decision node P_i contains a pointer to its left successor node P_{i_0} and one to its right successor node P_{i_1}. Garbled $\widetilde{P}_i$ contains encryptions of these pointers and of successors' respective keys $\Delta_{i_0}, \Delta_{i_1}$. Further, since we want to prevent the LBP evaluator from following both successor nodes, we additionally separately encrypt the data needed to decrypt P_{i_0} and P_{i_1} with random keys k_i^0 and k_i^1 respectively. Evaluator later will receive (one of) k_i^j, depending on his input (see block ObliviousLinearSelect), which will enable him to decrypt and follow only the corresponding successor node. The used *semantically secure symmetric encryption* scheme can be instantiated as $\mathsf{Enc}_k^s(m) = m \oplus H(k||s) = \mathsf{Dec}_k^s(m)$, where s is a unique identifier used once, and $H(k||s)$ is a pseudo-random function (PRF) evaluated on s and keyed with k, e.g., a cryptographic hash function from the SHA-2 family. In CreateGarbledLBP, we use the following technical improvement from [KJGB06]: Instead of encrypting twice (sequentially, with Δ_i and k_i^j), we encrypt successor P_{i_j}'s data with $\Delta_i \oplus k_i^j$. Each classification node is garbled simply by including its label directly into the parent's node (instead of the decryption key Δ_i). This eliminates the need for inclusion of classification nodes in the garbled LBP and increases the size of each garbled decision node by only two bits denoting the type of its successor nodes. This technical improvement allows to reduce the size of the garbled LBP by up to a factor of 2, depending on the number of classification nodes. Finally, the two successors' encryptions are randomly permuted.

We note that sometimes the order of nodes in a LBP may leak some information. To avoid this, in the garbling process we randomly permute the nodes of the LBP (which results in the corresponding substitutions

Algorithm 1 CreateGarbledLBP

Input $\mathcal{S}$**:** LBP $\mathcal{L} = \langle \{P_1, .., P_z\}, \mathit{Left}, \mathit{Right} \rangle$. For $i \leq d$, P_i is a decision node $\left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$. For $i > d$, P_i is a classification node $\langle c_i \rangle$.

Output $\mathcal{S}$**:** (i) Garbled LBP $\widetilde{\mathcal{L}} = \left\langle \{\widetilde{P}_1, .., \widetilde{P}_d\} \right\rangle$; (ii) Compl. garbled inputs $\widetilde{W}_1, .., \widetilde{W}_d$; (iii) Perm. lin. comb. vectors $\hat{\mathbf{a}}_\mathbf{1}^\ell, .., \hat{\mathbf{a}}_\mathbf{d}^\ell$; (iv) Perm. thresholds $\hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$

```
1:  choose a random permutation $\Pi$ of the set $1, .., d$ with $\Pi[1] = 1$.
2:  choose key $\Delta_1 := 0^t$, rand. keys $\Delta_i \in_R \{0,1\}^t$, $1 < i \leq d$ for enc. decision nodes
3:  for $i = 1$ to $d$ do $\{P_i = \left\langle \mathbf{a_i^\ell}, t_i^{\ell'} \right\rangle$ is a decision node$\}$
4:      let permuted index $\hat{i} := \Pi[i]$
5:      set perm. linear combination vector $\hat{\mathbf{a}}_{\hat{\mathbf{i}}}^\ell := \mathbf{a_i^\ell}$; perm. threshold value $\hat{t}_{\hat{i}}^{\ell'} := t_i^{\ell'}$
6:      choose rand. compl. garbled value $\widetilde{W}_{\hat{i}} = \langle \widetilde{w}_i^0 = \langle k_{\hat{i}}^0, \pi_{\hat{i}} \rangle, \widetilde{w}_i^1 = \langle k_{\hat{i}}^1, 1 - \pi_{\hat{i}} \rangle \rangle$
7:      let left successor $i_0 := \mathit{Left}[i]$, $\hat{i}_0 := \Pi[i_0]$ (permuted)
8:      if $i_0 \leq d$ then $\{P_{i_0}$ is a decision node$\}$
9:          let $m^{\hat{i},0} := \left\langle \texttt{"decision"}, \hat{i}_0, \Delta_{\hat{i}_0} \right\rangle$
10:     else $\{P_{i_0} = \langle c_{i_0} \rangle$ is a classification node$\}$
11:         let $m^{\hat{i},0} := \langle \texttt{"classification"}, c_{i_0} \rangle$
12:     end if
13:     let right successor $i_1 := \mathit{Right}[i]$, $\hat{i}_1 := \Pi[i_1]$ (permuted)
14:     if $i_1 \leq d$ then $\{P_{i_1}$ is a decision node$\}$
15:         let $m^{\hat{i},1} := \left\langle \texttt{"decision"}, \hat{i}_1, \Delta_{\hat{i}_1} \right\rangle$
16:     else $\{P_{i_1} = \langle c_{i_1} \rangle$ is a classification node$\}$
17:         let $m^{\hat{i},1} := \langle \texttt{"classification"}, c_{i_1} \rangle$
18:     end if
19:     let garbled decision node $\widetilde{P}_{\hat{i}} := \left\langle Enc_{k_{\hat{i}}^{\pi_{\hat{i}}} \oplus \Delta_{\hat{i}}}^{\hat{i},0}(m^{\hat{i},\pi_{\hat{i}}}), Enc_{k_{\hat{i}}^{1-\pi_{\hat{i}}} \oplus \Delta_{\hat{i}}}^{\hat{i},1}(m^{\hat{i},1-\pi_{\hat{i}}}) \right\rangle$
20: end for
21: return $\widetilde{\mathcal{L}} := \left\langle \{\widetilde{P}_1, .., \widetilde{P}_d\} \right\rangle; \widetilde{W}_1, .., \widetilde{W}_d; \hat{\mathbf{a}}_\mathbf{1}^\ell, .., \hat{\mathbf{a}}_\mathbf{d}^\ell; \hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$
```

in the encrypted pointers). The start node P_1 remains the first node in $\widetilde{\mathcal{L}}$. Additionally, garbled nodes are padded s.t. they all have the same size.

The output of CreateGarbledLBP is $\widetilde{\mathcal{L}}$ (to be sent to $\mathcal{C}$), and the randomness used in its construction (to be used by $\mathcal{S}$ in the next phase).

Complexity (cf. Table 2). $\widetilde{\mathcal{L}}$ contains d garbled nodes $\widetilde{P}_i$ consisting of two ciphertexts of size $\lceil \log d \rceil + t + 1$ bits each (assuming classification labels c_j have less bits than this). The size of $\widetilde{\mathcal{L}}$ is $2d(\lceil \log d \rceil + t + 1) \sim 2d(\log d + t)$ bits.

Tiny LBPs. In case of tiny LBPs with a small number of decision nodes d we describe an alternative construction method for garbled LBPs with asymptotic size $2^d \log(z - d)$ in §B.

Phase II: ObliviousLinearSelect. In this phase, $\mathcal{C}$ obliviously obtains the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ which correspond to the outcome of the comparison of the linear combination of the attribute vector with the threshold for each garbled node. These garbled values will then be used to evaluate the garbled LBP $\widetilde{\mathcal{L}}$ in the next phase.

In ObliviousLinearSelect, the input of $\mathcal{C}$ is the private attribute vector $\mathbf{x}^\ell$ and $\mathcal{S}$ inputs the private outputs of CreateGarbledLBP: complementary garbled values $\widetilde{W}_1 = \langle \widetilde{w}_1^0, \widetilde{w}_1^1 \rangle, .., \widetilde{W}_d = \langle \widetilde{w}_d^0, \widetilde{w}_d^1 \rangle$, permuted linear combination vectors $\hat{\mathbf{a}}_\mathbf{1}^\ell, .., \hat{\mathbf{a}}_\mathbf{d}^\ell$, and permuted threshold values $\hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$. Upon completion of the ObliviousLinearSelect protocol, $\mathcal{C}$ obtains the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$, as follows: if $\hat{\mathbf{a}}_\mathbf{i}^\ell \circ \mathbf{x}^\ell > \hat{t}_i^{\ell'}$, then $\widetilde{w}_i = \widetilde{w}_i^1$; else $\widetilde{w}_i = \widetilde{w}_i^0$. $\mathcal{S}$ learns nothing about $\mathcal{C}$'s inputs. We note that ObliviousLinearSelect can be viewed as an instance of *conditional oblivious transfer* [BK04].

We give two efficient instantiations for ObliviousLinearSelect in §3.4.

Phase III: EvalGarbledLBP. In the last phase, $\mathcal{C}$ receives the garbled LBP $\widetilde{\mathcal{L}}$ from $\mathcal{S}$, and evaluates it locally with algorithm EvalGarbledLBP as shown in Algorithm 2. This algorithm additionally gets the garbled values $\widetilde{w}_1, .., \widetilde{w}_d$ output by ObliviousLinearSelect as inputs and outputs the classification label $c = \mathcal{L}(\mathbf{x}^\ell)$.

Algorithm 2 EvalGarbledLBP

Input $\mathcal{C}$**:** (i) Garbled LBP $\widetilde{\mathcal{L}} = \left\langle \{\widetilde{P}_1, .., \widetilde{P}_d\} \right\rangle$; (ii) Garbled input values $\widetilde{w}_1, .., \widetilde{w}_d$

Output $\mathcal{C}$**:** Classification label c such that $c = \mathcal{L}(\mathbf{x}^\ell)$

```
1: let $\hat{i} := 1$; $\Delta_{\hat{i}} := 0^t$ (start at root)
2: while true do
3:   let $\langle k_{\hat{i}}, \pi_{\hat{i}} \rangle := \widetilde{w}_{\hat{i}}$; $\langle c^0_{\hat{i}}, c^1_{\hat{i}} \rangle := \widetilde{P}_{\hat{i}}$; $\langle \text{type}_{\hat{i}}, \text{data}_{\hat{i}} \rangle := \text{Dec}^{\hat{i}, \pi_{\hat{i}}}_{k_{\hat{i}} \oplus \Delta_{\hat{i}}}(c^{\pi}_{\hat{i}})$
4:   if $\text{type}_{\hat{i}}$ = "decision" then
5:     let $\left\langle \hat{i}, \Delta_{\hat{i}} \right\rangle := \text{data}_{\hat{i}}$
6:   else
7:     let $\langle c \rangle := \text{data}_{\hat{i}}$
8:     return $c$
9:   end if
10: end while
```

$\mathcal{C}$ traverses the garbled LBP $\widetilde{\mathcal{L}}$ by decrypting garbled decision nodes along the evaluation path starting at $\widetilde{P}_1$. At each node $\widetilde{P}_{\hat{i}}$,[4] $\mathcal{C}$ takes the garbled attribute value $\widetilde{w}_{\hat{i}} = \langle k_{\hat{i}}, \pi_{\hat{i}} \rangle$ together with the node-specific key $\Delta_{\hat{i}}$ to decrypt the information needed to continue evaluation of the garbled successor node until the correct classification label c is obtained.

It is easy to see that some information about $\mathcal{L}$ is leaked to $\mathcal{C}$, namely: (i) the total number d of *decision* nodes in the program $\widetilde{\mathcal{L}}$, and (ii) the length of the evaluation path, i.e., the number of decision nodes that have been evaluated before reaching the classification node. We note that in many cases this is acceptable. If not, this information can be hidden using appropriate padding of $\mathcal{L}$. We further note that $\widetilde{\mathcal{L}}$ cannot be reused. Each secure evaluation requires construction of a new garbled LBP.

3.4 Oblivious Linear Selection Protocol

We show how to instantiate the ObliviousLinearSelect protocol next.

A straight-forward instantiation can be obtained by evaluating a garbled *circuit* whose size depends on the number of attributes n. This construction is described in §A.1.

In the following, we concentrate on an alternative instantiation based on a *hybrid* combination of homomorphic encryption and garbled circuits which results in a better communication complexity.

Hybrid Instantiation. In this instantiation of ObliviousLinearSelect (see Fig. 3 for an overview), $\mathcal{C}$ generates a key-pair for the additively homomorphic encryption scheme and sends the public key $pk_{\mathcal{C}}$ together with the homomorphically encrypted attributes $[\![x^\ell_1]\!], .., [\![x^\ell_n]\!]$ to $\mathcal{S}$. Using the additively homomorphic property, $\mathcal{S}$ can compute the linear combination of these ciphertexts with the private coefficients $\hat{\mathbf{a}}^\ell_{\mathbf{i}}$ as $[\![y^{\ell'}_i]\!] := [\![\sum_{j=1}^n \hat{a}^\ell_{i,j} x^\ell_j]\!] = \prod_{j=1}^n [\![x^\ell_j]\!]^{\hat{a}^\ell_{i,j}}$, $1 \leq i \leq d$. Afterwards, the encrypted values $[\![y^{\ell'}_i]\!]$ are obliviously compared with the threshold values $\hat{t}^{\ell'}_i$ in the ObliviousParallelCmp protocol. This protocol allows $\mathcal{C}$ to obliviously obtain the garbled values corresponding to the comparison of $y^{\ell'}_i$ and $\hat{t}^{\ell'}_i$, i.e., $\widetilde{w}^0_i$ if $y^{\ell'}_i \leq \hat{t}^{\ell'}_i$ and $\widetilde{w}^1_i$ otherwise. ObliviousParallelCmp ensures that neither $\mathcal{C}$ nor $\mathcal{S}$ learns anything about the plaintexts $y^{\ell'}_i$ from which they could deduce information about the other party's private function or inputs.

[4] We use the permuted index $\hat{i}$ here to stress that $\mathcal{C}$ does not obtain any information from the order of garbled nodes.

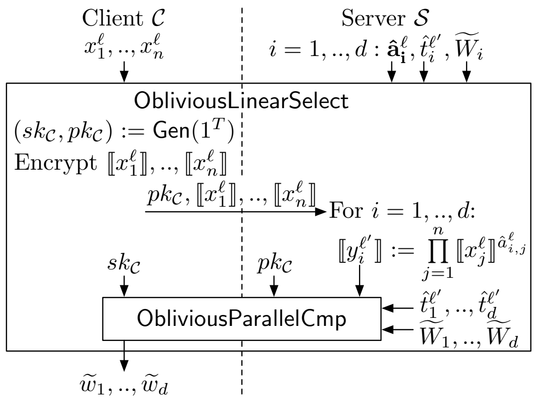
Client $\mathcal{C}$
$x_1^\ell, .., x_n^\ell$
Server $\mathcal{S}$
$i = 1, .., d : \hat{\mathbf{a}}_i^\ell, \hat{t}_i^{\ell'}, \widetilde{W}_i$
ObliviousLinearSelect
$(sk_\mathcal{C}, pk_\mathcal{C}) := \mathsf{Gen}(1^T)$
Encrypt $[\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
$pk_\mathcal{C}, [\![x_1^\ell]\!], .., [\![x_n^\ell]\!]$
For $i = 1, .., d$:
$[\![y_i^{\ell'}]\!] := \prod_{j=1}^{n} [\![x_j^\ell]\!]^{\hat{a}_{i,j}^\ell}$
$sk_\mathcal{C}$
$pk_\mathcal{C}$
ObliviousParallelCmp
$\hat{t}_1^{\ell'}, .., \hat{t}_d^{\ell'}$
$\widetilde{W}_1, .., \widetilde{W}_d$
$\widetilde{w}_1, .., \widetilde{w}_d$

Fig. 3. ObliviousLinearSelect - Hybrid

ObliviousParallelCmp *protocol (cf. Fig. 4).* The basic idea underlying this protocol is that $\mathcal{S}$ blinds the encrypted value $[\![y_i^{\ell'}]\!]$ in order to hide the encrypted plaintext from $\mathcal{C}$. To achieve this, $\mathcal{S}$ adds a randomly chosen value $R \in_R \mathbb{Z}_N$ [5] under encryption before sending them to $\mathcal{C}$ who can decrypt but does not learn the plain value. Afterwards, a garbled circuit C is evaluated which obliviously takes off the blinding value R and compares the result (which corresponds to $y_i^{\ell'}$) with the threshold value $t_i^{\ell'}$. We improve the communication complexity of this basic protocol which essentially corresponds to the protocol of [BPSW07] with the following two technical tricks:

Packing. Usually, the plaintext space of the homomorphic encryption scheme $\mathbb{Z}_N$ is substantially larger than the encrypted values $y_i^{\ell'}$. Hence, multiple encryptions, say d', can be packed together into one ciphertext before blinding and sending it to $\mathcal{C}$. This reduces the communication complexity and the number of decryptions that need to be performed by $\mathcal{C}$ by a factor of d'. For this, the encrypted values $-2^{\ell'-1} < y_i^{\ell'} < 2^{\ell'-1}$ are shifted into the positive range $(0, 2^{\ell'})$ first by adding $2^{\ell'-1}$ and afterwards are concatenated by computing $[\![y]\!] = [\![\sum_{i=1}^{d'} 2^{\ell'(i-1)}(y_i^{\ell'} + 2^{\ell'-1})]\!] = \prod_{i=1}^{d'}([\![2^{\ell'-1}]\!][\![y_i^{\ell'}]\!])^{2^{\ell'(i-1)}}$. The packed ciphertext $[\![y]\!]$ encrypts a $L' = d'\ell'$ bit value now.

Minimizing Circuit Size. As described before, the garbled circuit obliviously takes off the blinding value R. Instead of computing in the plaintext space of the homomorphic cryptosystem $\mathbb{Z}_N$ it is beneficial to compute over integers as circuit sizes are substantially smaller (inspired by the BITREP gate of [ST06]). For this, we ensure that no overflow occurs when blinding the L'-bit value y by adding $R \in_R \mathbb{Z}_N$. This can be achieved by choosing L' such that it is κ bits less than the bitlength of N, where κ is a statistical correctness parameter (e.g., $\kappa = 80$): $L' \leq T - \kappa$. Now, an overflow occurs only if the κ topmost bits of R are all ones which — as R was chosen randomly — happens with probability $2^{-\kappa}$ which is negligible in κ. The circuit subtracts the lowest L' bits of R from those of γ to obliviously reconstruct the value y before comparing this component-wise with the threshold values $t_i^{\ell'}$. The garbled circuit is naturally composed from subtraction and comparison circuits and has asymptotic size $12L't$ bits as described in §A.2.

If the negligible correctness error should be removed entirely, one could use a slightly larger circuit which checks if an overflow occurred and recovers the correct value. Alternatively, one could choose R from a smaller range which allows for possible attacks by a malicious $\mathcal{C}$ as described in §4.

Complexity (cf. Table 1). From $L' = d'\ell' \leq T - \kappa$ we can infer $d' = \frac{T-\kappa}{\ell'}$ as the maximum number of packed ciphertexts. As shown in Fig. 3, the ObliviousParallelCmp protocol needs to be run for d inputs which can be achieved by running the ObliviousParallelCmp protocol of Fig. 4 with d' inputs $\lceil \frac{d}{d'} \rceil$ times in parallel. The asymptotic communication complexity of the hybrid ObliviousLinearSelect protocol as shown in Table 1 consists of $n + \frac{d}{d'} = n + \frac{\ell'}{T-\kappa}d$ Paillier ciphertexts of size $2T$ bits each (HE), garbled circuits of size

[5] In contrast to [BPSW07], we choose R from the full plaintext space in order to protect against malicious behavior of $\mathcal{C}$ as explained in §4.

$\frac{d}{d'} \cdot 12L't = 12d\ell't$ bits (GC), and a $\frac{d}{d'} \cdot L' = d\ell'$ parallel OT protocol $\mathsf{OT}^{d\ell'}$ (OT). The number of moves is two for sending the homomorphic encryptions plus those of the underlying OT protocol ($\widetilde{C}$ can be sent with the last message of the OT protocol).

We note that further performance improvements can be achieved when the client only computes those values he will actually use in the LBP evaluation phase ("lazy evaluation"). All server-visible messages of OT must be performed to hide the evaluation path taken based on client's inputs.

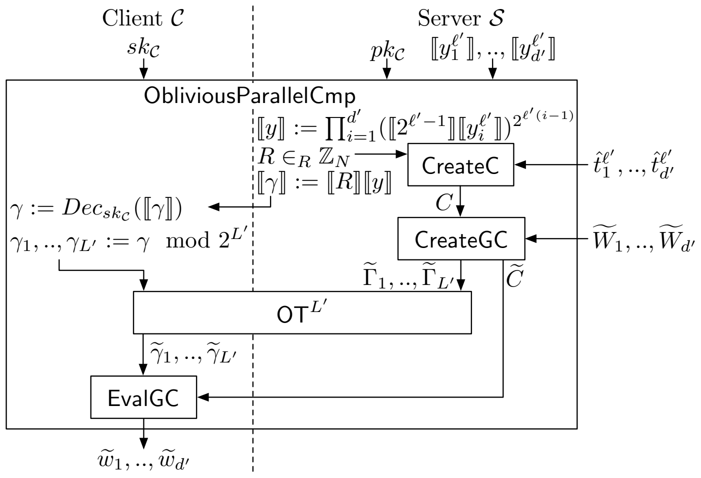
Client $\mathcal{C}$
$sk_{\mathcal{C}}$
Server $\mathcal{S}$
$pk_{\mathcal{C}}$
$[\![y_1^{\ell'}]\!], .., [\![y_{d'}^{\ell'}]\!]$
ObliviousParallelCmp
$[\![y]\!] := \prod_{i=1}^{d'}([\![2^{\ell'-1}]\!][\![y_i^{\ell'}]\!])^{2^{\ell'(i-1)}}$
$R \in_R \mathbb{Z}_N$
$[\![\gamma]\!] := [\![R]\!][\![y]\!]$
CreateC
$\hat{t}_1^{\ell'}, .., \hat{t}_{d'}^{\ell'}$
C
$\gamma := Dec_{sk_{\mathcal{C}}}([\![\gamma]\!])$
$\gamma_1, .., \gamma_{L'} := \gamma \mod 2^{L'}$
CreateGC
$\widetilde{W}_1, .., \widetilde{W}_{d'}$
$\widetilde{\Gamma}_1, .., \widetilde{\Gamma}_{L'}$
$\widetilde{C}$
$\mathsf{OT}^{L'}$
$\widetilde{\gamma}_1, .., \widetilde{\gamma}_{L'}$
EvalGC
$\widetilde{w}_1, .., \widetilde{w}_{d'}$

Fig. 4. ObliviousParallelCmp

Extension of [BPSW07] to LBPs. Our hybrid instantiation of the ObliviousLinearSelect protocol is a generalization of the ObliviousAttributeSelection protocol proposed in [BPSW07]. The protocol for secure evaluation of private BPs of [BPSW07] can easily be extended to a protocol for secure evaluation of private LBPs by computing a linear combination of the ciphertexts instead of obliviously selecting one ciphertext. We call this protocol "ext. [BPSW07]". However, our hybrid protocol is more efficient than ext. [BPSW07] as shown in the following.

3.5 Performance Improvements Over Existing Solutions

On the one hand, our protocols for secure evaluation of private LBPs extend the functionality that can be evaluated securely from OBDDs [KJGB06], private OBDDs [Sch08], and private BPs [BPSW07] to the larger class of private LBPs. On the other hand, our protocols can be seen as general protocols which simply become improved (more efficient) versions of the protocols of [KJGB06,Sch08,BPSW07] when instantiated for the respective special case functionality.

In the following, we summarize the employed techniques and the resulting performance improvements of our protocols over previous solutions (see Table 1 and Table 2). We stress that our improvements, achieved by combining our new technical tricks and some previously known techniques, are at the conceptual protocol level and will translate directly into any implementation, i.e., are independent of implementation details such as programming languages or hardware used.

Incorporate classification nodes into decision nodes: In contrast to previous protocols given in [KJGB06,Sch08,BPSW07], the size of our improved method for constructing garbled LBPs (cf. Algorithm 1 & 2 in §3.3) depends on the number of decision nodes d only which is smaller than the total number of nodes z (cf. Table 2), e.g., a full classification tree has $z = 2d+1$ nodes. Hence, $\mathcal{C}$ cannot infer the number of classification nodes in our protocols. As additionally the number of attributes for which the ObliviousLinearSelect protocol needs to be run is reduced from z down to d, the communication complexity of our hybrid protocol —

except sending the encrypted attributes (HE) — grows linearly in d instead of z (cf. rows "[BPSW07]"/"ext. [BPSW07]" vs. "Hybrid" in Table 1). For full classification trees this results in an overall improvement by a factor of two.

Tiny LBPs: For tiny LBPs with $d \leq 10$ decision nodes, e.g., the privacy-preserving remote diagnostics examples for iptables and mpg321 given in [BPSW07], or the medical ECG classification application described in §5, our alternative method for garbled LBPs described in §B results in a substantially smaller communication complexity than existing solutions (cf. Table 2).

Point-and-permute: The protocols of [KJGB06,BPSW07] use the semantically secure encryption scheme with 'special' properties of [LP04] where trial-decryptions of all entries must be performed until the 'right' entry was found (which can be decided using the special properties). To achieve the special properties, the ciphertext size is increased by κ (the statistical correctness parameter) additional bits [LP04]. In contrast to this, our protocols use a point-and-permute technique as described in §2.1 to directly identify the right entry to decrypt during evaluation of the GC as well as for the garbled LBP. This reduces the computation complexity for $\mathcal{C}$ as no trial-decryptions need to be performed. Also a standard semantically secure encryption scheme without padding can be used in our protocols which results in reduced communication complexity (cf. dependance on κ in column "GC" of Table 1 and column "Size of Garbled LBP" of Table 2).

Key-offsets: In the protocol of [BPSW07], the garbled circuits are encrypted symmetrically with a node-specific key. During evaluation, $\mathcal{C}$ obtains this key, decrypts the GC, and evaluates it. In contrast to this, our protocols use node-specific key-offsets Δ_i as described in §3.3 which result in lower computation complexity for both, $\mathcal{S}$ and $\mathcal{C}$ (adapted from [KJGB06]).

Packing: The hybrid ObliviousLinearSelect protocol improves over the ObliviousAttributeSelection protocol of [BPSW07] by packing together multiple ciphertexts before sending them back to $\mathcal{C}$ as described in §3.4. This reduces the communication complexity and the number of public-key decryptions that $\mathcal{C}$ needs to perform by a factor of $d' = \frac{T-\kappa}{\ell'}$ (cf. column HE in Table 1). For the privacy-preserving diagnostics application scenario given in [BPSW07] with $\ell' = 32, \kappa = 80$ and the minimum asymmetric security parameter $T = 1248$ recommended today [GQ09] this results in an improvement by another factor of $d' = 36$. (Packing together multiple ciphertexts into one was exploited before for example in [BPB08].)

Oblivious Selection Protocol	Private Function	Moves	Asymptotic Communication Complexity		
			GC	OT	HE
[BPSW07]	BP	OT + 2	$12z\ell(t+\kappa)$	$\mathrm{OT}_t^{z\ell}$	$(n+z)2T$
ext. [BPSW07] (§3.4)	LBP		$12z\ell'(t+\kappa)$	$\mathrm{OT}_t^{z\ell'}$	
our Hybrid (§3.4)	BP	OT + 2	$12d\ell t$	$\mathrm{OT}_t^{d\ell}$	$(n+\frac{\ell}{T-\kappa}d)2T$
	LBP		$12d\ell' t$	$\mathrm{OT}_t^{d\ell'}$	$(n+\frac{\ell'}{T-\kappa}d)2T$
our Circuit (§A.1)	BP	OT	$4(n\log d + 3d\log d)\ell t$	$\mathrm{OT}_t^{n\ell}$	
	LBP		$16nd(\ell^2+\ell')t$		

Table 1. Protocols for Secure Evaluation of Private BPs/LBPs with parameters z: #nodes, d: #decision nodes, n: #attributes, ℓ: bitlength of attributes, ℓ': bitlength of thresholds (for LBPs), t: symmetric security parameter, T: asymmetric security parameter, κ: statistical correctness parameter.

3.6 Correctness and Security Properties

As previously mentioned, protocol SecureEvalPrivateLBP securely and correctly evaluates private LBP in the semi-honest model. We formally state and prove the corresponding theorems in Appendix C.

Algorithm to Create/Evaluate Garbled LBP	Size of Garbled LBP in bit	Examples from [BPSW07] with $t = 80, \kappa = 80$			
		iptables $d = 4, z = 9$	mpg321 $d = 5, z = 9$	ECG classification (§5) $d = 6, z = 12$	nfs $d = 12, z = 17$
[KJGB06,BPSW07]	$2z(\lceil \log z \rceil + t + \kappa)$	$2,952$ bit	$2,952$ bit	$3,936$ bit	$5,610$ bit
Algorithm 1 & 2 (§3.3)	$2d(\lceil \log d \rceil + t + 1)$	664 bit	840 bit	$1,008$ bit	$2,040$ bit
Tiny GLBP (§B)	$2^d \lceil \log(z - d) \rceil$	48 bit	64 bit	192 bit	$12,288$ bit

Table 2. Algorithms to Create/Evaluate Garbled LBPs with parameters z: #nodes, d: #decision nodes, t: symmetric security parameter, κ: statistical correctness parameter.

Extensions to Malicious Players. We note that our protocols, although proven secure against semi-honest players, tolerate many malicious client behaviors. For example, many efficient OT protocols are secure against malicious chooser, and a malicious client is unable to tamper with the GC evaluation procedure. Further, our protocols can be modified to achieve full security in the malicious model. One classical way is to prove in zero-knowledge the validity of every step a party takes. However, this approach is far inefficient. We achieve malicious security simply by employing efficient sub-protocols proven secure against malicious players. (This is the transformation approach suggested in [BPSW07].) More specifically, we use committed OT, secure two-party computation on committed inputs, and verifiable homomorphic encryption schemes (see [JS07] for more detailed description).

Learning the LBP. A general problem of trying to hide a private function are so-called oracle attacks. Here, the attacker, having black-box access to the functionality, queries the functionality adaptively on multiple inputs and interpolates the private function from the results.[6]

As oracle attacks only use the black-box functionality, they can be applied to any secure implementation of a cryptographic protocol which realizes the black-box functionality — recall, the security definition only guarantees that the protocol does not reveal more information than what can be learned from the outputs in the plain algorithm. A possible solution to protect against this possibility to learn the LBP by successive queries is to limit the number of allowed accesses to the server.

Especially piecewise linear functions such as LBPs can be learned easily: It is easy to see that the boundaries of the decision regions produced by an LBP algorithm are piecewise linear in an n-dimensional space. If an attacker is able to find n points on each of the hyperplanes defining the boundary of the decision regions, then he is able to completely reveal the LBP algorithm. If the attacker can query the LBP protocol without any limits, finding such n points is relatively easy by using a bisection algorithm. The attacker starts with two attribute vectors yielding different classification results, then he applies a bisection algorithm on the line passing for such two vectors until he finds a point that is on (or very close) to the decision boundary. The overall complexity of this attack is only linear in n and hence it represents a serious threat.

4 A Technical Omission in [BPSW07] w.r.t. Malicious Client

In this section, we briefly present and fix a small technical omission, which led to an incorrect claim of security in the setting with semi-honest server and malicious client in [BPSW07, Section 4.4] (and indirectly propagated to [BS09]). Recall, the protocol of [BPSW07] is similar in the structure to our protocol. The problem appears in the ObliviousAttributeSelection subroutine, which is similar to (actually is a special case of) our ObliviousLinearSelect subroutine. The issue is that, for efficiency, [BPSW07] mask the $\mathcal{C}$-encrypted attribute values with relatively short random strings, before returning them back to $\mathcal{C}$. In the semi-honest model this guarantees that $\mathcal{C}$ is not able to match the returned strings to the attribute values he earlier sent, and the security of the entire protocol holds. However, the security breaks in case of a malicious $\mathcal{C}$. Indeed, such a $\mathcal{C}$ can send $\mathcal{S}$ very large values x_i, wait for the blinded responses and match these with the original x_i, allowing $\mathcal{C}$ to determine which of the attributes are used for the computation. (Indeed, whereas the lower bits are blinded correctly, the upper bits of the maliciously chosen large x_i remain the same.) We further

[6] Oracle attacks are used for example in image watermarking applications where it is referred to as sensitivity attack [KLvD98,CPFPG06].

note that malicious $\mathcal{C}$ will not even be caught since he will recover the blinding values and will be able to continue execution with his real inputs, if he wishes.

This attack can be prevented by choosing R randomly from the full plaintext domain $\mathbb{Z}_N$ instead (as done in our ObliviousParallelCmp protocol). With this modification, the blinded value is entirely random in $\mathbb{Z}_N$ and a malicious $\mathcal{C}$ cannot infer any information from it.

5 Application: Secure Classification of Medical Data

In this section we present our motivating example application for secure evaluation of private LBPs – privacy-preserving classification of biomedical data. While one might need to protect a variety of types of biomedical data (e.g., [BCA$^+$93,TWBT95,FLK08,KTB$^+$03]) we consider privacy-preserving classification of ElectroCardioGram (ECG) signals as a simple *representative* example. A patient (client $\mathcal{C}$) owns an ECG signal and asks a service provider (server $\mathcal{S}$) to determine which class the ECG signal belongs to. $\mathcal{C}$ requires $\mathcal{S}$ to gain no knowledge about the ECG signal (as this is sensitive personal data of $\mathcal{C}$), whereas $\mathcal{S}$ requires no disclosure of details of the classification algorithm to $\mathcal{C}$ (as this represents valuable intellectual property of $\mathcal{S}$). To achieve this, we map the signal processing chain of an established ECG classification algorithm [ASSK07,GSK02] to secure evaluation of a private LBP, as described next (cf. Fig. 5).

Inputs and Outputs. The classification algorithm of [ASSK07] and [GSK02] classifies the ECG trace of a single heartbeat into one of 6 classes: Normal Sinus Rhythm (NSR), Atrial Premature Contraction (APC), Premature Ventricular Contraction (PVC), Ventricular Fibrillation (VF), Ventricular Tachycardia (VT), and SupraVentricular Tachycardia (SVT).

Signal Preprocessing. Following the structure of the classification algorithm described in [ASSK07], each ECG trace is modeled by a 4-th order autoregressive (AR) model first (cf. [BJR76] for details). The AR model parameters and a proper set of statistics of the prediction error are merged together to obtain the so-called feature vector (AR Modeling and Feature Extraction). This results in the 21-element-long feature vector which is quantized to ℓ-bit signed integers (Quantization) – the attribute vector $\mathbf{x}^\ell = \{x_1^\ell, .., x_{21}^\ell\}$ which is input into the secure classification as follows.

Secure Classification. Secure classification consists of three steps: First, the quadratic discriminant function (QDF) classifier (cf. [MWI92] for details) projects $\mathbf{x}^\ell$ onto 6 dimensions $\mathbf{a}_\mathbf{i}^\ell, i = 1, .., 6$. Afterwards, the signs of the projections are extracted (Sign Extraction) and finally used to traverse a classification tree (Classification). It is easy to see that these three steps of the signal processing chain exactly correspond to the evaluation of an LBP with $\mathbf{x}^\ell$ as attribute vector and 6 nodes $P_i = \left\langle \mathbf{a}_\mathbf{i}^\ell, 0 \right\rangle, i = 1, .., 6$ as shown in Fig. 6. (Note, the coefficients of the linear combination vector $\mathbf{a}_\mathbf{i}^\ell$ represent the knowledge embedded within the classifier and are usually computed by relying on a set of training ECG's [ASSK07].) This LBP can be efficiently and securely evaluated using protocols presented in this paper.

6 Implementation of Classification Algorithm and Communication Complexity

We have implemented and tested the classification algorithm for ECG data using LBPs to work in the plain domain. To experimentally evaluate the dependence of the classification accuracy, we built a dataset of 1200 ECG signals taken from the MIT-BIH Arrhythmia, Malignant Ventricular Arrhythmia and Supraventricular Arrhythmia Databases available in PhysioBank archives [GAG$^+$00]. The dataset contained 200 examples of each of the 6 classes considered by the classifier. We split the dataset into two parts: 60 signals of each class for training the classifier and 140 for testing it. In figure 7 the classification rate is plotted as a function of the attribute length ℓ. An attribute length of $\ell = 44$ bit results in a reasonable classification accuracy of 88.6%.

The estimated communication complexity of the protocol for secure classification of ECG data with parameters $z = 12, d = 6, n = 21, \ell = 44, \ell' \overset{(1)}{=} 92, \kappa = 80$ is given in Fig. 8. This table depicts different sizes for current security parameter recommendations for short-term ($t = 80, T = 1248$), medium-term ($t =$

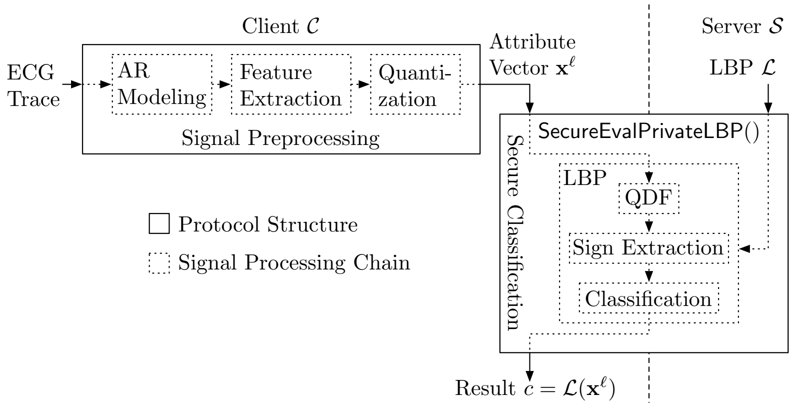

Fig. 5. Secure ECG Classification with LBP

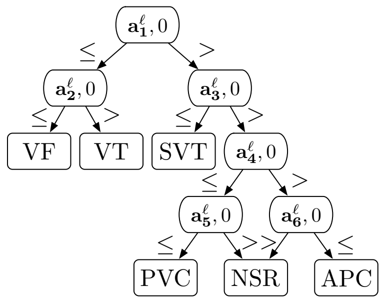

Fig. 6. LBP for ECG Classification

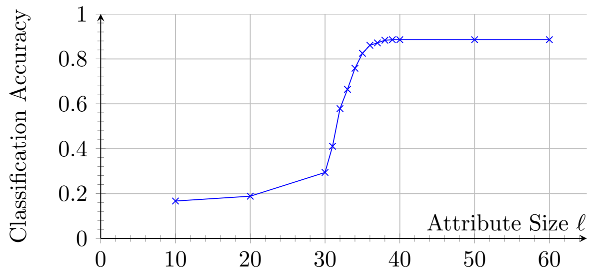

Fig. 7. Classification Accuracy over Attribute Size ℓ

Protocol	ext. [BPSW07] (§3.4)	Hybrid (§3.4)	Circuit (§A.1)
Moves	4	4	2
Security Level	Asymptotic Communication Complexity		
short-term	329 kByte	101 kByte	39.0 MByte
medium-term	411 kByte	143 kByte	54.7 MByte
long-term	453 kByte	164 kByte	62.5 MByte

Fig. 8. Estimated Communication Complexity

$112, T = 2432$) and long-term ($t = 128, T = 3248$) security [GQ09]. We use the parallel version of the two-move OT protocol of [NP01] which can be implemented over a suitably chosen curve, e.g., curve secp160r1 for $t = 80$, secp224r1 for $t = 112$, resp. secp256r1 for $t = 128$ from the SECG standard [SEC00b,SEC00a,Bro05].

To evaluate the real communication and computation complexity of the Hybrid and the GC instantiation of the protocol, we implemented both protocols in C++ using the Miracl library [Sco09]. The measured communication and computation complexities for short-term security are shown in Table 3. The tests were performed on two PCs with 3 GHz Intel Core Duo processor and 4GB memory connected via Gigabit Ethernet. The measured communication complexities are slightly higher than the estimated ones shown in Table 8 as communication is byte-oriented.

Protocol Type	Communication Client		Computation [s] Client		Server	
	sent	received	cpu	total	cpu	total
Hybrid (§3.4)	17.7 kByte	94.5 kByte	2.0	29.0	4.8	27.6
Circuit (§A.1)	19.0 kByte	40.6 MByte	4.7	48.5	11.5	48.8

Table 3. Performance of our protocols for secure ECG classification

Acknowledgments. We thank anonymous reviewers of ESORICS 2009 for their helpful comments.

References

AIR01. W. Aiello, Y. Ishai, and O. Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT'01*, volume 2045 of *LNCS*, pages 119–135. Springer, 2001.

ASSK07. U. R. Acharya, J. Suri, J. A. E. Spaan, and S. M. Krishnan. *Advances in Cardiac Signal Processing*, chapter 8. Springer, 2007.

BCA$^+$93. F. M. Bennett, D. J. Christini, H. Ahmed, K. Lutchen, J. M. Hausdorff, and N. Oriol. Time series modeling of heart rate dynamics. In *Computers in Cardiology 1993. Proceedings.*, pages 273–276, 1993.

BFK$^+$09. M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, LNCS. Springer, 2009. Full version available at `http://eprint.iacr.org/2009/195`.

BJR76. G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis.* Holden-day San Francisco, 1976.

BK04. I. F. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Advances in Cryptology – ASIACRYPT'04*, volume 3329 of *LNCS*, pages 515–529. Springer, 2004.

BPB08. T. Bianchi, A. Piva, and M. Barni. Efficient pointwise and blockwise encrypted operations. In *ACM workshop on Multimedia and security (MM&SEC'08)*, pages 85–90. ACM, 2008.

BPSW07. J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *ACM Conference on Computer and Communications Security (CCS'07)*, pages 498–507. ACM, 2007.

Bro05. D. R. L. Brown. Certicom proposal to revise SEC 1: Elliptic curve cryptography, version 1.0. Technical report, Certicom Research, 2005. Available from `http://www.secg.org`.

BS09. J. Brickell and V. Shmatikov. Privacy-preserving classifier learning. In *Financial Cryptography and Data Security (FC'09)*, LNCS. Springer, 2009.

CL01. Y.-C. Chang and C.-J. Lu. Oblivious polynomial evaluation and oblivious neural learning. In *Advances in Cryptology – ASIACRYPT'01*, volume 2248 of *LNCS*, pages 369–384. Springer, 2001.

CPFPG06. P. Comesana, L. Perez-Freire, and F. Perez-Gonzalez. Blind newton sensitivity attack. In *IEE Proceedings on Information Security*, volume 153, pages 115–125, 2006.

DCDZ05. S. J. Delany, P. Cunningham, D. Doyle, and A. Zamolotskikh. Generating estimates of classification confidence for a case-based spam filter. In *Case-Based Reasoning Research and Development*, volume 3620 of *LNCS*, pages 177–190. Springer, 2005.

DJ01. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public-Key Cryptography (PKC'01)*, LNCS, pages 119–136. Springer, 2001.

FLK08. P. Flor-Henry, J. L. Lind, and Z. J. Koles. Quantitative EEG and source localization in fibromyalgia. *International Journal of Psychophysiology*, 69(3):142–142, 2008.

GAG$^+$00. A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, June 2000.

Gol04. O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2004.

Goo09. Google Health, 2009. `https://www.google.com/health`.

GQ09. D. Giry and J.-J. Quisquater. Cryptographic key length recommendation, March 2009. `http://keylength.com`.

GSK02. D. F. Ge, N. Srinivasan, and S. M. Krishnan. Cardiac arrhythmia classification using autoregressive modeling. *BioMedical Engineering OnLine*, 1(1):5, 2002.

HRD$^+$07. J. Ha, C. J. Rossbach, J. V. Davis, I. Roy, H. E. Ramadan, D. E. Porter, D. L. Chen, and E. Witchel. Improved error reporting for software that uses black-box components. In *Programming Language Design and Implementation (PLDI'07)*. ACM, 2007.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*. Springer, 2003.

IP07. Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography (TCC'07)*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.

JS07. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology – EUROCRYPT'07*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.

Kil88. J. Kilian. Founding cryptography on oblivious transfer. In *ACM Symposium on Theory of Computing (STOC'88)*, pages 20–31. ACM, 1988.

KJGB06. L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM Conference on Computer and Communications Security (CCS'06)*, pages 410–420. ACM Press, 2006.

KLvD98. T. Kalker, J.-P. M. G. Linnartz, and M. van Dijk. Watermark estimation through detector analysis. In *IEEE International Conference on Image Processing (ICIP'98)*, volume 1, pages 425–429. IEEE, 1998.

KS08a. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP'08)*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

KS08b. V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC'08)*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008.

KTB$^+$03. P. Kalra, J. Togami, G. Bansal, A. W. Partin, M. K. Brawer, R. J. Babaian, L. S. Ross, and C. S. Niederberger. A neurocomputational model for prostate carcinoma detection. *Cancer*, 98(9):1849–1854, 2003.

Lip03. H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology – ASIACRYPT'03*, volume 2894 of *LNCS*. Springer, 2003.

Lip08. H. Lipmaa. Private branching programs: On communication-efficient cryptocomputing. Cryptology ePrint Archive, Report 2008/107, 2008. `http://eprint.iacr.org/`.

LP04. Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. ECCC Report TR04-063, Electronic Colloq. on Comp. Complexity, 2004.

MNPS04. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX*, 2004. `http://www.cs.huji.ac.il/project/Fairplay`.

MWI92. G. J. McLachlan, J. Wiley, and W. InterScience. *Discriminant analysis and statistical pattern recognition*. Wiley New York, 1992.

NP01. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.

OPB07. C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *European Journal of Information Systems (EURASIP)*, 2007(1):1–10, 2007.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

PCB$^+$08. A. Piva, M. Caini, T. Bianchi, C. Orlandi, and M. Barni. Enhancing privacy in remote data classification. *New Approaches for Security, Privacy and Trust in Complex Environments (SEC'08)*, 2008.

Pin02. B. Pinkas. Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explor. Newsl.*, 4(2):12–19, 2002.

PSS09. A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *Applied Cryptography and Network Security (ACNS'09)*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.

RGI05. J. Rodriguez, A. Goni, and A. Illarramendi. Real-time classification of ECGs on a PDA. *IEEE Transact. on Inform. Technology in Biomedicine*, 9(1):23–34, 2005.

Sch08. T. Schneider. Practical secure function evaluation. Master's thesis, University of Erlangen-Nuremberg, February 27, 2008.

Sco09. M. Scott. Multiprecision integer and rational arithmetic c/c++ library, March 2009. `http://shamus.ie`.

SEC00a. Standards for efficient cryptography, SEC 1: Elliptic curve cryptography. Technical report, Certicom Research, 2000. Available from `http://www.secg.org`.

SEC00b. Standards for efficient cryptography, SEC 2: Recommended elliptic curve domain parameters. Technical report, Certicom Research, 2000. Available from `http://www.secg.org`.

SS08. A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *International Conference on Information Security and Cryptology (ICISC'08)*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

ST06. B. Schoenmakers and P. Tuyls. Efficient binary conversion for paillier encrypted values. In *Advances in Cryptology – EUROCRYPT'06*, volume 4004 of *LNCS*, pages 522–537. Springer, 2006.

SYY99. T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *IEEE Symp. on Found. of Comp. Science (FOCS'99)*, pages 554–566. IEEE, 1999.

TWBT95. S. Todd, M. Woodward, C. Bolton-Smith, and H. Tunstall-Pedoe. An investigation of the relationship between antioxidant vitamin intake and coronary heart disease in men and women using discriminant analysis. *Journal of Clinical Epidemiology*, 48(2):297–305, 1995.

Yao86. A. C. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.

A Oblivious Linear Selection Protocol

A.1 Circuit Instantiation

ObliviousLinearSelect can be instantiated based on the secure evaluation of a garbled circuit as shown in Fig. 9. First, $\mathcal{S}$ creates a boolean circuit C with ℓ-bit inputs $x_1^\ell,..,x_n^\ell$ and output bits $w_1,\dots,w_d$ that obliviously computes the intended functionality as described below. The circuit is evaluated securely with Yao's garbled circuit protocol, i.e., $\mathcal{S}$ creates a garbled circuit $\widetilde{C}$ (using algorithm CreateGC) which is sent to $\mathcal{C}$ along with the garbled inputs corresponding to $\mathcal{C}$'s inputs $x_1^\ell,..,x_n^\ell$ in a $\mathsf{OT}^{n\ell}$ protocol, and finally $\mathcal{C}$ evaluates $\widetilde{C}$ on these garbled inputs (using algorithm EvalGC) to obtain the garbled output values $\widetilde{w}_1,\dots,\widetilde{w}_d$.

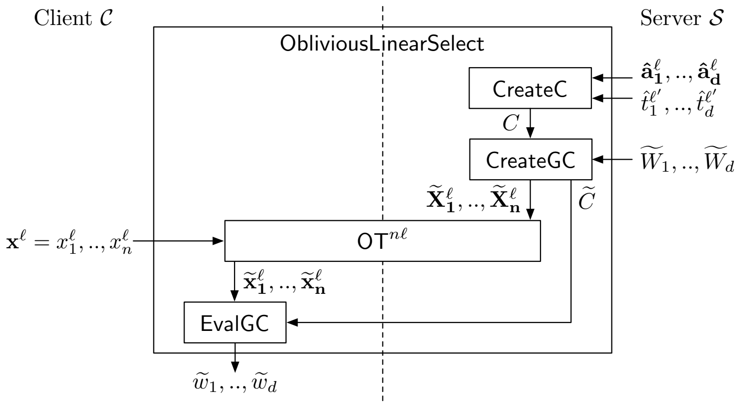
Client $\mathcal{C}$
Server $\mathcal{S}$
ObliviousLinearSelect
CreateC
$\hat{\mathbf{a}}_1^\ell,..,\hat{\mathbf{a}}_d^\ell$
$\hat{t}_1^{\ell'},..,\hat{t}_d^{\ell'}$
C
CreateGC
$\widetilde{W}_1,..,\widetilde{W}_d$
$\widetilde{\mathbf{X}}_1^\ell,..,\widetilde{\mathbf{X}}_n^\ell$
$\widetilde{C}$
$\mathbf{x}^\ell = x_1^\ell,..,x_n^\ell$
$\mathsf{OT}^{n\ell}$
$\widetilde{\mathbf{x}}_1^\ell,..,\widetilde{\mathbf{x}}_n^\ell$
EvalGC
$\widetilde{w}_1,..,\widetilde{w}_d$

Fig. 9. ObliviousLinearSelect - Circuit Instantiation

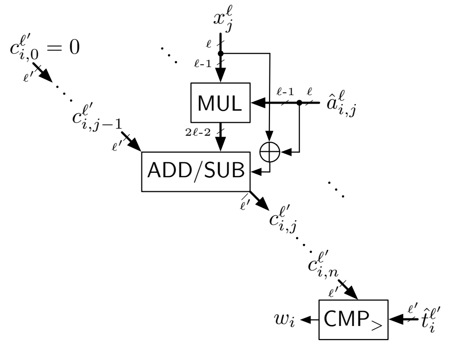

Fig. 10. Linear Selection Circuit C

LBP. In case of LBPs, the circuit C needs to compute $w_i = (\hat{\mathbf{a}}_i^\ell \circ \mathbf{x}^\ell > \hat{t}_i^{\ell'}) = (\sum_{j=1}^n \hat{a}_{i,j}^\ell \cdot x_j^\ell > \hat{t}_i^{\ell'})$, $1 \le i \le d$. As shown in Fig. 10, an efficient circuit construction can be obtained by first multiplying the magnitudes of x_j^ℓ and $\hat{a}_{i,j}^\ell$ with an unsigned integer multiplication circuit (MUL). Afterwards, the sign is determined by combining the sign bits of x_j^ℓ and $\hat{a}_{i,j}^\ell$ with an XOR gate ($\oplus$). Depending on this sign, the multiplied value is added or subtracted from the intermediate result $c_{i,j-1}^{\ell'}$ with an integer addition/subtraction circuit (ADD/SUB). Hence, the intermediate values $c_{i,J}^{\ell'}$ carry the sum of the first J summands, i.e., $c_{i,J}^{\ell'} = \sum_{j=1}^J \hat{a}_{i,j}^\ell \cdot x_j^\ell$. In the end, the final value $c_{i,n}^{\ell'} = \sum_{j=1}^n \hat{a}_{i,j}^\ell \cdot x_j^\ell$ is compared with the threshold value $\hat{t}_i^{\ell'}$ using an integer comparison circuit (CMP$_>$). The circuit can be generated automatically with the compiler of [PSS09] into a garbled circuit of size $|\widetilde{C}| = d(n(|\mathsf{MUL}| + |\oplus| + |\mathsf{ADD/SUB}|) + |\mathsf{CMP}_>|) = d(n([16(\ell-1)^2 - 16(\ell-1)+4] + [4] + [16\ell']) + 4\ell')(t+1) \sim 16tnd(\ell^2+\ell')$ bit.

BP. In case of BPs, a substantially smaller circuit C' can be constructed to compute the functionality $w_i = (x_{\hat{\alpha}_i} > \hat{t}_i^\ell)$. This circuit first obliviously selects the input $x_{\hat{\alpha}_i}^\ell$ from the inputs $x_1^\ell,..,x_n^\ell$. This can be achieved by using selection blocks S_d^n as follows: An S_d^n selection block is a circuit which can obliviously select for each of its d outputs any of its n inputs. By using ℓ such selection blocks in parallel (one for each bit of the ℓ bits), the circuit can obliviously select $x_{\hat{\alpha}_1}^\ell,..,x_{\hat{\alpha}_d}^\ell$ from $x_1^\ell,..,x_n^\ell$. Afterwards, the selected values are compared with the respective threshold value $\hat{t}_i^\ell$ using an integer comparison circuit (CMP$_>$). Using efficient selection block constructions of [KS08b] together with the comparison blocks implemented in [PSS09] this results in a garbled circuit of size $|\widetilde{C}'| = \ell|S_d^n| + d|\mathsf{CMP}_>| = (\ell[4(n+3d)\lceil\log d\rceil + 4n - 16d + 12] + d[4\ell])(t+1) \sim 4(n\log d + 3d\log d)\ell t$ bit.

Complexity (cf. Table 1). The circuit-based instantiation of ObliviousLinearSelect needs the same number of moves as the underlying OT protocol as the garbled circuit can be sent with the last message of the OT protocol. The asymptotic communication complexity of the circuit-based ObliviousLinearSelect protocol as shown in Table 1 is that of the $\mathsf{OT}^{n\ell}$ protocol (OT) plus the size of the garbled circuit given above (GC).

Security. The circuit-based instantiation of ObliviousLinearSelect is secure against malicious $\mathcal{C}$ and semi-honest $\mathcal{S}$ in the standard model. This follows directly from the corresponding security of Yao's garbled circuit [LP04].

A.2 Hybrid Instantiation - Technical Details

In the hybrid instantiation of the ObliviousLinearSelect protocol described in §3.4, $\mathcal{S}$ constructs a circuit C with L' inputs corresponding to $\gamma \mod L'$ and k outputs $w_1, .., w_{d'}$: First, the L' least significant bits of R are subtracted from the inputs corresponding to $\gamma \mod L'$ resulting in $y = \bar{y}_{d'}^{\ell'} || \ldots || \bar{y}_1^{\ell'}$ with an integer subtraction circuit (SUB). Then, each $\bar{y}_i^{\ell'}$, $1 \leq i \leq d$ is compared with its corresponding threshold $\hat{t}_i^{\ell'}$ with an integer comparison circuit ($\mathsf{CMP}_>$). The circuit can be generated automatically with the compiler of [PSS09] into a garbled circuit of size $|\widetilde{C}| = |SUB| + d'|\mathsf{CMP}_>| = (8L' + d'(4\ell'))(t+1) \sim 12L't$ bit.

B Garbled LBP Construction for Tiny LBPs

As alternative to the improved method of [BPSW07] described in §3.3 (Algorithm 1 and Algorithm 2), the garbled LBP can be constructed using a single Yao gate with d inputs as described next. The garbled LBP $\widetilde{\mathcal{L}}$ needs to obliviously map the garbled inputs $\widetilde{w}_1, .., \widetilde{w}_d$ to the corresponding classification label c as explained in §3.2. This can trivially be implemented with a Yao gate with d inputs which encrypts for each of the 2^d possible input combinations the index of the corresponding label. As the total number of classification nodes is $z - d$, their index can be encoded with $\lceil \log(z - d) \rceil$ bits. Hence, the overall size of the garbled gate is $|\widetilde{\mathcal{L}}'| = 2^d \lceil \log(z - d) \rceil \sim 2^d \log(z - d)$ bits. As the size of this alternative construction for garbled LBPs grows exponentially in d, this is feasible for tiny LBPs only. While – in contrast to the method described in §3.3 – this method reveals the number of classification nodes and their labels but hides the length of the evaluation path without need for padding with dummy decision nodes.

C Security and Correctness Claims and Proofs

Correctness of SecureEvalPrivateLBP is easiest to verify by considering its components. By correctness, we mean the requirement that, upon completion of the protocol execution, two honest parties produce the correct output.

Firstly, observe that both our proposed implementations of ObliviousLinearSelect are indeed correct. Verification of this is somewhat tedious, but straightforward. The circuit-based protocol is based on Yao's GC construction, and its correctness verification is similar to that of GC. Our hybrid ObliviousLinearSelect algorithm additionally makes use of homomorphic encryption; the correctness of this amendment is easily seen as well.

Further, given the correctness of ObliviousLinearSelect, the CreateGarbledLBP/EvalGarbledLBP pair allows to correctly evaluate $\widetilde{\mathcal{L}}$. This can be easily verified through the method of construction and blind evaluation of the garbled LBP $\widetilde{\mathcal{L}}$. Indeed, EvalGarbledLBP simply unravels the encryptions and permutations added by CreateGarbledLBP. Since ObliviousLinearSelect correctly provides the decryption keys corresponding to the parties' inputs, $\mathcal{C}$ obtains the correct classification label $c = \mathcal{L}(\mathbf{x}^{\ell})$.

Security Proofs. We now prove the security properties of our constructions. All of the proofs are standard, and we omit the most tedious details. However, full proofs are readily obtained from our presentation. We show security in the semi-honest setting, with respect to standard simulation-based definitions (e.g., Goldreich [Gol04]). When we say "protocol π is secure", we mean "protocol π securely implements the (implied) ideal functionality". We explicate this statement for our main Theorem 3. We first show security of both ObliviousLinearSelect variants.

Theorem 1 (ObliviousLinearSelect — **Circuit-Based).** *The circuit-based* ObliviousLinearSelect *protocol is secure against semi-honest adversaries.*

Proof. We first note that the circuit C that $\mathcal{S}$ construct is independent of the parties' inputs, and is public. Further, standard GC construction and execution using a secure OT protocol (cf. §2.1) is secure in the semi-honest model. We refer the reader to multiple existing constructions and proofs in the literature (e.g., [LP04,PSS09]). This completes the proof of the theorem. □

Theorem 2 (ObliviousLinearSelect — Hybrid). *The hybrid* ObliviousLinearSelect *protocol is secure against semi-honest adversaries.*

Proof. The proof of security is simple, but somewhat tedious; it is similar to corresponding proofs in [BPSW07]. At the high level, $\mathcal{C}$'s security follows from the fact that $\mathcal{C}$ only sends encrypted data, and from the security of employed OT. $\mathcal{S}$'s security follows from the method of construction of the garbled circuits and the homomorphic manipulations, and from the security of employed OT. □

Let $\mathcal{L}$ be a LBP, as defined in Definition 1. We are interested in secure evaluation of the following functionality, where d is the total number of *decision* nodes of $\mathcal{L}$, and e is the the number of nodes that have been evaluated before reaching a classification node (depth of evaluation path).

Functionality 1

$$f_{LBPEval}(\mathbf{x}^\ell, \mathcal{L}) = ((\mathcal{L}(\mathbf{x}^\ell), d, e), \textit{empty string})) \tag{2}$$

We note that we allow $\mathcal{C}$ to learn d and e, since hiding these values carries high cost, and for the simplicity of presentation. If desired to keep these values private, dummy nodes can be inserted in the LBP $\mathcal{L}$ to pad and thus partially hide both d and e. This is a standard simple technique, and we do not further discuss it. Our proofs can be trivially modified to accommodate this addition.

Theorem 3 (Security). *Let* ObliviousLinearSelect *be secure in the semi-honest model (cf. Theorems 1 and 2). Then Protocol* SecureEvalPrivateLBP *evaluates Functionality 1 securely in the semi-honest model.*

Proof. The proof of Theorem 3 is similar to existing proofs of security of GC and OBDD, such as those of [LP04,KJGB06]. At the highest level, $\mathcal{C}$'s security follows from the security of the underlying OT. Indeed, $\mathcal{C}$ does not send anything other than the messages of the OT protocol. $\mathcal{S}$'s security follows from the security of OT, as well as from the method of garbling the LBP $\mathcal{L}$, allowing $\mathcal{C}$ to follow a single evaluation path. Further, the structure of the LBP remains hidden from $\mathcal{C}$ since the node pointers are encrypted.

Formal proof of security requires construction of simulators Sim_S and Sim_C, which produce a view indistinguishable from the views $\mathcal{S}$ and $\mathcal{C}$ have respectively. Recall, a semi-honest party's view consists of its input, used randomness, and the messages it received.

First consider the case that $\mathcal{S}$ is corrupt. We construct a simulator Sim_S that, given the input $\mathcal{L}$, and the output $\perp$, simulates VIEW_S. Sim_S uses the simulator guaranteed by the secure ObliviousLinearSelect as follows. Sim_S, given the input $\mathcal{L}$, runs CreateGarbledLBP, and feeds the appropriate output of CreateGarbledLBP into the simulator of ObliviousLinearSelect. Sim_S's output consists of Sim_S' input $\mathcal{L}$, the randomness used to run CreateGarbledLBP, and whatever ObliviousLinearSelect's simulator output. It is easy to see that this simulation is indistinguishable from the real view VIEW_S (where the indistinguishability is computational (resp. statistical) if the output of ObliviousLinearSelect's simulator is computationally (resp. statistically) close to its respective real view).

Now consider the case that $\mathcal{C}$ is corrupt. This case is a little more involved, but is standard nonetheless. We construct a simulator Sim_C that, given the input $\mathbf{x}^\ell$, and the output $\mathcal{L}(\mathbf{x}^\ell)$, simulates VIEW_S. Notice that, in particular, $\mathcal{C}$ expects to receive a garbled LBP $\widetilde{\mathcal{L}}$, together with the wire values that correspond to $\mathcal{C}$'s inputs. That is, when evaluated, $\widetilde{\mathcal{L}}$ should produce $\mathcal{L}(\mathbf{x}^\ell)$. The main simulation challenge is that Sim_C does not know $\mathcal{L}$, and thus cannot simply honestly generate $\widetilde{\mathcal{L}}$ by running CreateGarbledLBP. We show how to generate a fake garbled $\widetilde{\mathcal{L}}'$ that evaluates to $\mathcal{L}(\mathbf{x}^\ell)$, yet is indistinguishable from a real $\widetilde{\mathcal{L}}$. Once this is accomplished, the remainder of the simulation is easy (calling simulator of ObliviousLinearSelect and plugging its output).

Recall, we allow $\mathcal{C}$ to learn the size (more specifically, the number of decision nodes) d of $\mathcal{L}$, and the depth e of the evaluation path. Therefore, this information is made available to Sim_C. Sim_C then constructs $\widetilde{\mathcal{L}}'$ that always evaluates to $\mathcal{L}(\mathbf{x}^\ell)$, regardless of what $\mathcal{L}$ actually does. Sim_C builds it by first generating, a (garbled) evaluation path of length e. Each path node is generated according to Algorithm 1, with the following exceptions. First, the path terminates with the encrypted node containing two classification labels – encryptions of $\mathcal{L}(\mathbf{x}^\ell)$. Second, each path node's encrypted pointers point to the same (next) node on the path.

Once the path is generated, Sim_C generates a number of fake nodes (so that the total number of nodes is d). These nodes are properly formatted random encryptions of random values – they will never be reached by the evaluation procedure, and they are indistinguishable from the real nodes. Finally, we note that Sim_C actually places the nodes in random order and adjusts the pointers of the nodes that lie on the evaluation path accordingly. This completes the description of Sim_C.

It is not hard to see that the view generated by Sim_C is computationally indistinguishable from the real execution. This is done using a standard hybrid argument over the nodes of LBP, similarly to the proof presented in [KJGB06]. As in [KJGB06], ultimately, the hybrid argument boils down to the indistinguishability of the real node, and the simulated node, containing two encryptions (with different keys) of identical values. We observe that if the encryption *Enc* is semantically secure, or if *Enc* is a PRFG, then the "non-revealed" encryptions of the two nodes are indistinguishable. Looking at it from another angle, a distinguisher that tells apart real and simulated nodes, can be used to construct a distinguisher for PRFG (or semantically-secure encryption).

Finally, Sim_C uses the constructed simulated LBP $\widetilde{\mathcal{L}}'$ to provide inputs to the simulator of the view of $\mathcal{C}$ of ObliviousLinearSelect. For each node i, Sim_C randomly chooses a bit b_i, and uses $w_i^{b_i}$ as the secret to be output by ObliviousLinearSelect. Then the output of the simulator is plugged into the output of Sim_C.

This completes the outline of the proof. $\square$